New Rewritings and Optimizations for Regular Path Queries

Gösta Grahne and Alex Thomo

Concordia University,
Montreal, Canada
{grahne, thomo}@cs.concordia.ca

Abstract. All the languages for querying semistructured data and the web use as an integral part regular expressions. Based on practical observations, finding the paths that satisfy those regular expressions is very expensive. In this paper, we introduce the "maximal partial rewritings" (MPR's) for regular path queries using views. The MPR's are always exact and more useful for the optimization of the regular path queries than other rewritings from previously known methods. We develop an algorithm for computing MPR's and prove, through a complexity theoretic analysis, that our algorithm is essentially optimal. Also, we present query answering algorithms that utilize exact partial rewritings for regular path queries and conjunctive regular path queries respectively.

1 Introduction

Semistructured data are self-describing collections, whose structure can naturally model irregularities that cannot be captured by relational or object-oriented data models [1]. This kind of data is usually best formalized in terms of labeled graphs, where the graphs represent data found in many useful applications such as web information systems, XML data repositories, digital libraries, communication networks, and so on. Virtually, all the query languages for semi-structured data provide the possibility for the user to query the database through regular expressions. The design of query languages using regular path expressions, is based on the observation that many of the recursive queries that arise in practice amount to graph traversals. These queries are in essence graph patterns, and the answers to the query are subgraphs of the database that match the given pattern [16, 7, 3, 4].

For example, for answering a query containing in it the regular expression $(\_^* \cdot \mathit{article}) \cdot (\_^* \cdot \mathit{ref} \cdot \_^* \cdot (\mathit{abiteboul} + \mathit{vianu}))$, one should find all the paths having at some point an edge labeled *article*, followed by any number of other edges, then by an edge *ref*, and finally by an edge labeled with *abiteboul* or *vianu*.

Based on practical observations, the most expensive part of answering queries on semistructured data is to find those graph patterns described by regular expressions. This is because a regular expression can describe arbitrarily long paths in the database, which means in turn an arbitrary number of physical accesses. Hence, it is clear that having a good optimizer for answering regular path queries is very important. This optimizer can be used for the broader classes of query languages for semistructured data, as we illustrate in the paper.

In semistructured data as well as in other data models, such as relational and "object oriented," the importance of utilizing views is well recognized [15, 3, 14, 17]. Simply stated the problem is: Given a query Q and a set of views $\{V_1, \ldots, V_n\}$, find a representation (rewriting) of Q by means of the views and then answer the query on the basis of this representation.

The most notable methods for obtaining rewritings for regular path queries are by Calvanese *et al* in [3, 5, 6]. However, these methods rewrite –using views– only complete words of the query. But in practice, the cases in which we can infer from the views full words for the query, are very "ideal." The views can cover *partial* words that can be satisfactorily long for using them in optimization, but if they are not complete words, they are ignored by the above mentioned methods. It would however be desirable to have a partial rewriting in order to capture and utilize all the information provided by the views.

The problem of computing a partial rewriting is initially treated by Calvanese *et at* in [3]. There this problem is considered as an extension of the complete rewriting, enriching the set of the views with new elementary one-symbol views, chosen among the database relations (or symbols). The choice of the new elementary views is done in a brute force way, using a cost criterion depending on the application. However, there are cases when the algorithm of [3] for computing partial rewritings gives "too much" (in a sense to be made precise later). It essentially contains redundant un-rewritten (sub)words from the query.

In a previous paper [10], the present authors presented another algorithm for computing contained partial rewritings. Using that algorithm we avoid getting "too much," but unfortunately, the rewriting is not guaranteed to be exact, which diminishes its usability.

In this paper we will introduce the "maximal partial rewritings" (MPR's) for regular path queries using views. We will show that they don't give "too much," that they are always exact and more useful for the optimization of the regular path queries.

Then, we will present an algorithm that, when using exact partial rewritings, optimizes the evaluation of regular path queries to a database. We also explore the use of the partial rewritings for the optimization of the wider class of conjunctive regular path queries (CRPQ's). We introduce the "conjunctive exact partial rewritings" (CEPR's) and present an algorithm for utilizing them in CRPQ evaluation.

Finally, through a complexity theoretic analysis, we prove that our algorithm for computing the "maximal partial rewritings" is essentially optimal.

2 Background

Semistructured databases. We consider a database to be an edge labeled graph. This graph model is typical in semistructured data, where the nodes of the database graph represent the objects and the edges represent the attributes of the objects, or relationships between the objects.

Formally, we assume that we have a universe of objects D and a finite alphabet Δ, called the *database alphabet*. Objects of D will be denoted a, b, c, a', b', $\ldots, a_1, b_2, \ldots$, etc. Elements of Δ will be denoted $R, S, T, R', S', \ldots, R_1, S_1, \ldots$, etc.

A *database* DB over (D, Δ) is a pair (N, E), where $N \subseteq D$ is a set of nodes and $E \subseteq N \times \Delta \times N$ is a set of directed edges labeled with symbols from Δ.

Queries and rewritings of queries using views. A *regular path query* Q is a finite or infinite regular language over the alphabet Δ. Let Q be a regular path query, and $DB = (N, E)$ a database. Then the *answer to* Q on DB is defined as:

$$ans(Q, DB) = \{(a, b) \in N^2 \ : \ a \xrightarrow{W} b \text{ for some } W \in Q\}.$$

Let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of *view definitions* with each V_i being a finite or infinite regular language over Δ. Associated with each view definition V_i there is a view name v_i. We call the set $\Omega = \{v_1, \ldots, v_n\}$ the *outer* or *view alphabet*. For each $v_i \in \Omega$, we set $def(v_i) = V_i$. The substitution *def* associates with each view name v_i, in the Ω alphabet, the language V_i. Also, we extend the substitution *def* to the Δ alphabet associating each symbol with itself. The substitution *def* is applied to words, languages, and regular expressions, over $\Omega \cup \Delta$ in the usual way (see e. g. [12]). Sometimes we need to refer to regular expressions representing the languages Q and V_i. In order to simplify the notation, we will blur the distinction between the regular expressions and the languages that they represent.

The *maximally contained rewriting*, of a user query Q using $\mathbf{V}$ ($MCR_{\mathbf{V}}(Q)$), is the language Q' over Ω that includes *all* the words $v_{i_1} \ldots v_{i_k} \in \Omega^*$, such that

$$def(v_{i_1} \ldots v_{i_k}) \subseteq Q.$$ [1]

The $MCR_{\mathbf{V}}(Q)$ can be empty even if the desired answer is not. Suppose, for example, that the query Q is $Q = R_1 \ldots R_{100}$ and we have available two views V_1 and V_2, where $V_1 = R_1 \ldots R_{49}$ and $V_2 = R_{51} \ldots R_{100}$. It is easy to see that $MCR_{\mathbf{V}}(Q)$ is empty. However, depending on the application, a "partial rewriting" such as $v_1 R_{50} v_2$ could be useful. In the next section we develop a formal algebraic framework for the partial rewritings. In sections 5 and 6 we demonstrate the usefulness of the partial rewritings in query optimization.

3 Partial Rewritings

Let L be a language and M an ϵ-free language, both of them over some alphabet Δ. Let m be a symbol outside Δ and set $def(m) = M$. A *partial M-rewriting* of

[1] It is easy to see that this definition of $MCR_{\mathbf{V}}(Q)$ matches exactly the language of the output automaton from Theorem 1 in [3], which is the rewriting as defined there, with respect to the view alphabet.

L is a language L' over $\Delta \cup \{m\}$, such that $def(L') \subseteq L$. The partial rewriting is said to be *exact*, if $def(L') = L$. Since the (complete) rewritings, i.e. the ones that use only the m symbol, are special cases of the partial rewritings, we will blur the distinction between them and call the partial rewritings just *rewritings*.

It is clear that there can be several M-rewritings of the same language L. In order to compare different rewritings, we introduce a partial order between the languages over $\Delta \cup \{m\}$. With this partial order we want to capture the intuition that the more subwords on the Δ-alphabet that have been replaced by m in a rewriting, the "bigger" (and "better") the rewriting.

Let L_1 and L_2 be languages over $\Delta \cup \{m\}$. We define L_1 to be M-smaller than L_2, denoted $L_1 \leq_M L_2$, if it is possible to substitute by m some (not necessarily all) occurrences of words over M occurring as subwords in L_1, and obtain L_2 as a result.

Obviously $\leq_M$ is transitive and reflexive. It is not antisymmetric, as for instance $\{mRR, mm, RRRR\} \leq_M \{mm, RRRR\}$, and $\{mRR, mm, RRRR\}$ $\geq_M \{mm, RRRR\}$, when M is for example $\{RR\}$. However, if we define $L_1 \equiv_M L_2$ iff $L_1 \leq_M L_2$ and $L_2 \leq_M L_1$, we will get a partial order on the equivalence classes.

Notably, we have that if a set L' is $\leq_M$-maximal, then its equivalence class is singleton. To show this, we first present the following theorem.

Theorem 1. *Let L' be a language on $\Delta \cup \{m\}$. Then L' is $\leq_M$-maximal, if and only if, there does not exist a word $W \in L'$, such that $W = W_1W_2W_3$, and $W_2 \in M$.*

Proof. "if." This direction is easy to see because, if there is no word of M that appears as a subword in any of the words of L', then it is impossible to obtain any new $\leq_M$-larger language by substituting some subword belonging to M with m.

"only if." Let's suppose that there are subwords of L', which belong to M. Then, for each word $W \in L'$ compute a word W_M by exhaustively replacing occurrences of M words in W until nothing can be replaced anymore. If there is no subword in W that can be replaced by m, then compute W_M equal with W. Clearly, for each word W there is at least one such word, and the number of steps for computing it is bounded by the length of W. Now, consider the language $L'' = \bigcup_{W \in L'} \{W_M\}$. For the language L'' we have that (a) $L'' \neq L'$, (b) $L' \leq_M L''$, and (c) we cannot obtain any new language from L'' by substituting m in place of some subword belonging to M. We can see that from (a) and (c), L' and L'' cannot belong to the same equivalence class, and from (b), L'' is $\leq_M$-larger than L'. All the above show that L' cannot be maximal, and this is a contradiction. $\square$

Corollary 1. *Let L' be a language on $\Delta \cup \{m\}$. If L' is $\leq_M$-maximal, then its $\equiv_M$-equivalence class is a singleton.*

Proof. Since L' is $\leq_M$-maximal, from the above theorem we have that from L' we cannot obtain any new language by substituting m in place of some subword belonging to M. This means in turn, that the equivalence class of L' is a singleton. $\square$

Now, consider this partial order restricted to the set of all the M-rewritings of a language L. We will denote the restricted partial order with $\leq_M^L$. Obviously, we are interested in the $\leq_M^L$-maximal M-rewritings of L. With a similar reasoning as before, we can prove the next theorem and corollary. The main difference is that we cannot now replace just any subword which is a word in M. Naturally, a word $W = W_1W_2W_3$, where $W_2 \in M$ and $def(W_1mW_3) \subseteq L$, is not yet an "optimal" word, and we call W_2 a subword *eligible for replacement*.

Theorem 2. *Let L' be a rewriting of L on $\Delta \cup \{m\}$. Then L' is $\leq_M^L$-maximal, if and only if, there does not exist a word $W \in L'$, such that $W = W_1W_2W_3$, where $W_2 \in M$, and $def(W_1mW_3) \subseteq L$.*

Proof. "if." This direction is easy to see because, if there is no word of M that appears as a subword eligible for replacement in any of the words of L', then it is impossible to obtain any new ($\leq_M^L$-larger) rewriting from L'.

"only if." Let's suppose that there are subwords eligible for replacement in the words of L'. Then, for each word $W \in L'$ compute a word W_M, by exhaustively replacing the eligible for replacement subwords in W, until nothing can be replaced anymore. If there is no eligible for replacement subword in W, then compute W_M equal with W. Clearly, for each word W there is at least one such word W_M, and the number of steps for computing it, is bounded on the length of W. Now, consider the rewriting $L'' = \bigcup_{W \in L'}\{W_M\}$. For the rewriting L'' we have that (a) $L'' \neq L'$, (b) $L' \leq_M^L L''$, and (c) we cannot obtain any new $\leq_M^L$-larger rewriting from L''. We can see that from (a) and (c), L' and L'' cannot belong to the same equivalence class, and from (b), L'' is $\leq_M^L$-larger than L'. All the above show that L' cannot be a maximal rewriting, and this is a contradiction. $\square$

Corollary 2. *Let L' be a rewriting on $\Delta \cup \{m\}$. If L' is $\leq_M^L$-maximal, then its $\equiv_M^L$-equivalence class is a singleton.*

Proof. Since L' is $\leq_M^L$-maximal, from the above theorem we have that from L' we cannot obtain any new $\leq_M^L$-larger rewriting. This means in turn that the equivalence class of L' is a singleton. $\square$

Also, we require exactness for a rewriting. Only then a rewriting becomes useful for query optimization [4, 11].

A rewriting that is both $\leq_M^L$-maximal and exact is the union of all $\leq_M^L$-maximal M-rewritings of L. We call this rewriting the *maximal partial M-rewriting* of L, and denote it with $MPR_M(L)$.

From all the above, we can see that the $MPR_M(L)$ is the set of *all* the words on $\Delta \cup \{m\}$ with no subword for potential "contained replacement."

Example 1. Let $M = \{RSR, S\}$, and

$$L = \{RSRS, SRSR, RSRRSR, SS, SRSRS, SSS\}.$$

Then

$$MPR_M(L) = \{mm, SmS\}.$$

Here, mm is obtained from $RSRS$, $SRSR$, $RSRRSR$, or SS. The word SmS is obtained from $SRSRS$ or SSS. There are no more eligible subwords for replacement. For instance, replacing subwords in $\underline{S}m\underline{S}$ that belong to M, would violate the containment condition necessary for being a rewriting.

Formally, we have that:

Theorem 3. *The rewriting $MPR_M(L)$ is $\leq_M^L$-maximal and exact.*

Proof. The $\leq_M^L$-maximality follows from the fact that the $MPR_M(L)$ is the union of the sets of words W with no subwords eligible for replacement. For the exactness, observe that by the definition we have $def(MPR_M(L)) \subseteq L$. On the other hand, consider a word $W \in L$. Testing a finite number of times (function of the length of W) we can find a word $W' \in (\Delta \cup \{m\})^*$ with the smallest possible number of Δ symbols, and such that $W \in def(W') \subseteq L$. Clearly, $W' \in MPR_M(L)$. $\square$

We can give a natural generalization of the definition of the maximal partial M-rewriting for the case when we like to replace subwords not from one language only, but from a finite set of languages (such as a finite set of view definitions). For this purpose, suppose we are given the ϵ-free view languages $\mathbf{V} = \{V_1, \ldots, V_n\}$ and a target query language Q, all of them over the alphabet Δ. Also, consider the view alphabet $\Omega = \{v_1, \ldots, v_n\}$, for which as defined in Section 2, we have $def(v_i) = V_i$, for $i \in [1, n]$. A *partial* $\mathbf{V}$*-rewriting* of Q is a language Q' over $\Delta \cup \Omega$, such that $def(Q') \subseteq Q$. The partial rewriting is said to be *exact* if $def(Q') = Q$. Then, in order to compare the rewriting we define analogously the partial order $\leq_{\mathbf{V}}$ on $(\Delta \cup \Omega)^*$ as follows.

Let Q_1 and Q_2 be languages over $\Delta \cup \Omega$. We define Q_1 to be $\mathbf{V}$-smaller than Q_2, denoted $Q_1 \leq_{\mathbf{V}} Q_2$, if it is possible to substitute by $v_{i_1}, \ldots, v_{i_k}$ some (not necessarily all) occurrences of words over $V_{i_1}, \ldots, V_{i_k}$ respectively, occurring as subwords in Q_1, and obtain Q_2 as a result.

As before, we restrict this partial order to the set of all the $\mathbf{V}$-rewritings of a language Q. Similarly, we denote the restricted partial order with $\leq_{\mathbf{V}}^Q$. Finally, we define the *maximal partial* $\mathbf{V}$*-rewriting* of Q, denoted $MPR_{\mathbf{V}}(Q)$, to be the *union* of all $\leq_{\mathbf{V}}^Q$-maximal $\mathbf{V}$-rewritings Q' of Q. Clearly, as in Theorem 3, we can show that the $MPR_{\mathbf{V}}(Q)$ is $\leq_{\mathbf{V}}^Q$-maximal and exact.

We will compare in the following the MPR's with the partial rewritings proposed in the literature. We begin with the partial rewriting of [3]. In that paper, candidate sub-alphabets $\Delta' \subseteq \Delta$ are selected using some cost criteria, and then the symbols of Δ' are considered as new elementary one-symbol views. Note that there are exponentially many candidate sub-alphabets. Each time, using the algorithm presented in [3], a rewriting is computed and after that its exactness is tested. However, the algorithm used, although very suitable for computing all the rewritings in the view alphabet Ω, can compute "non-optimal" $\Delta \cup \Omega$ words when it is used for computing partial rewritings. As an example, consider the regular path query $Q = R_1R_2 + R_2^5$ and two views $V_1 = R_1$ and $V_2 = R_2^5$. Then, the algorithm of [3] will give as partial rewriting $V_1R_2 + V_2 + R_2^5$, which has the redundant "non-optimal" word R_2^5. As a consequence, the partial rewriting of [3] is not always $\leq_{\mathbf{V}}^{Q}$-maximal. We call the partial rewriting of [3] the *maximally contained partial rewriting* or $MCPR_{\mathbf{V}}$ because it contains *all* the words W over $\Delta' \cup \Omega$, such that $def(W) \subseteq Q$. Observe here that the maximality in this rewriting is with respect to the containment of $\Delta' \cup \Omega$ words and not really to their optimality regarding the non-view symbols.

Now, let's consider the partial rewriting presented in [11]. This rewriting is the union of all $\leq_{\mathbf{V}}$-maximal $\mathbf{V}$-rewritings of Q. By Theorem 1 we have that this is the set of all "mixed" words W on the alphabet $\Omega \cup \Delta$ with no subword in $V_1 \cup \cdots \cup V_n$, such that their substitution by *def* is contained in the query Q. We call this set *exhaustive lower partial rewriting*, or $ELPR_{\mathbf{V}}(Q)$, because we replace subwords of Q with view symbols thinking only about the optimality of words with regard to the non-view symbols, but not caring about the exactness of the rewriting. Clearly, since any $\leq_{\mathbf{V}}$-maximal rewriting is also $\leq_{\mathbf{V}}^{Q}$-maximal, the result is that we get always a $\leq_{\mathbf{V}}^{Q}$-maximal rewriting, but its exactness is not guaranteed.

Finally, $MPR_{\mathbf{V}}(Q)$ is a rewriting that satisfies both the optimality with regard to the non-view symbols and the exactness. Summarizing, and also including the maximally contained (complete) rewriting $MCR_{\mathbf{V}}$ in the comparison, we have the following table.

	$\leq_{\mathbf{V}}^{Q}$-maximality	Exactness
MCR [3]	YES	NO
$MCPR$ [3]	NO	YES
$ELPR$ [11]	YES	NO
MPR	YES	YES

4 Computing the Maximal Partial Rewriting

To this end, we will first give a characterization of the maximal partial M-rewriting of a language L. The construction in the proof of our characterization provides the basic algorithm for computing the rewriting $MPR_M(L)$ on a given regular language L.

The construction is based on finite transducers. A *finite transducer* $T = (S, I, O, \delta, s_0, F)$ consists of a finite set of states S, an input alphabet I, and an output alphabet O, a starting state s_0, a set of final states F, and a transition-output relation $\delta \subseteq S \times I^* \times S \times O^*$. Intuitively, for instance $(s_0, U, s_1, W) \in \delta$ means that if the transducer is in state s_0 and reads word U it can go to state s_1 and emit the word W. For a given word $U \in I^*$, we say that a word $W \in O^*$ is an *output of* T *for* U if there exists a sequence $(s_0, U_1, s_1, W_1) \in \delta$, $(s_1, U_2, s_2, W_2) \in \delta$, ..., $(s_{n-1}, U_n, s_n, W_n) \in \delta$ of state transitions in T, such that $s_n \in F$, $U = U_1 \ldots U_n$, and $W = W_1 \ldots W_n$. We write $W \in T(U)$, where

$T(U)$ denotes set of all outputs of T for the input word U. For a language $L \subseteq I^*$ we define $T(L) = \bigcup_{U \in L} T(U)$. It is well known that $T(L)$ is regular whenever L is.

We are now in a position to state our characterization theorem.

Theorem 4. *Let L and M be regular languages over an alphabet Δ. There exists a finite transducer T and a regular language M', such that*

$$MPR_M(L) = (T(L^c))^c \cap M',$$

where $(.)^c$ denotes set complement.

Proof. Let $A = (S, \Delta, \delta, s_0, F)$ be a nondeterministic finite automaton that accepts the language M. Let us consider the finite transducer:

$$T = (S \cup \{s_0'\}, \Delta, \Delta \cup \{m\}, \delta', s_0', \{s_0'\}),$$

where δ', written as a relation, is

$$\begin{aligned}\delta' = &\{(s, R, s', \epsilon) : (s, R, s') \in \delta\} \cup \\ &\{(s_0', R, s_0', R) : R \in \Delta\} \cup \\ &\{(s_0', R, s, \epsilon) : (s_0, R, s) \in \delta\} \cup \\ &\{(s_0', R, s_0', m) : (s_0, R, s) \in \delta \text{ and } s \in F\} \cup \\ &\{(s, R, s_0', m) : (s, R, s') \in \delta \text{ and } s' \in F\}.\end{aligned}$$

Intuitively, transitions in the first set of δ' are the transitions of the "old" automaton, modified so as to produce ϵ as output. Transitions in the second set mean that "if we like, we can leave everything unchanged," i.e. each symbol gives itself as output. Transitions in the third set are for jumping non-deterministically from the new initial state s_0' to the states of the old automaton A, that are reachable in one step from the old initial state s_0. These transitions give ϵ as output. Transitions in the fourth set are for handling special cases, when from the old initial state s_0, an old final state can be reached in one step. In these cases we can replace the one symbol words accepted by A with the special symbol m. Finally, the transitions of the fifth set are the most significant. Their meaning is: in a state, where the old automaton has a transition by a symbol, say R, to an old final state, there will be, in the transducer, an additional transition R/m to s_0', which is also the (only) final state of T. Observe that, if the transducer T decides to leave the state s_0' while a suffix U of the input string is unscanned, and enter the old automaton A, then it can return back only if there is a prefix U' of U, such that $U' \in L(A)$. In such a case, the transducer T replaces U' with the special symbol m.

Given a word of $W \in \Delta^*$ as input, the finite transducer T replaces arbitrarily many occurrences of words of M in W with the special symbol m. For an example, suppose M is given by the automaton in Figure 1, top. The corresponding finite transducer is shown in the same figure, bottom. It consists of the automaton for M, whose transitions now produce as output ϵ, plus the state s_0', and the additional transitions, which are drawn with dashed arrows.

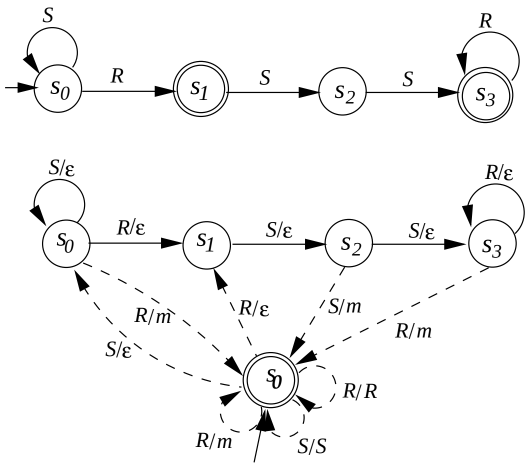

Fig. 1. An example of the construction for a replacement transducer

If L' is a language on Δ, it is straightforward to verify that

$$\begin{aligned} T(L') = L' \cup \{&U_1 m U_2 m \dots m U_k : \\ &\text{for some } U \in L' \text{ and words } W_i \in M, \\ &U = U_1 W_1 U_2 W_2 \dots W_{k-1} U_k\}. \end{aligned}$$

Recall that, we have $def(m) = M$, and $def(R) = R$ for each $R \in \Delta$. From the above, we can also easily characterize the set $T(L')$ from another point of view. Namely, $T(L')$ is the set of all words W on $\Delta \cup \{m\}$, such that $def(W) \cap L' \neq \emptyset$.

Now, let's consider the transduction $T(L^c)$. As characterized above, $T(L^c)$ will be the set of all words W on $\Delta \cup \{m\}$ such that $def(W) \cap L^c \neq \emptyset$. Hence, $T(L^c))^c$, being the complement of this set, will contain *all* the $\Delta \cup \{m\}$ words, such that *all* the Δ-words in their substitution by *def* will be contained in L. This is the containment condition for a word on $\Delta \cup \{m\}$ to be in a rewriting. Clearly, $T(L^c))^c$ is a rewriting, and namely, it is the union of *all* the M-rewritings of L. Hence, $MPR_M(L) \subseteq T(L^c))^c$.

However, in order to compute $MPR_M(L)$, what we like is not a contained arbitrary, but a contained exhaustive replacement of words from M. To achieve this goal, we should filter out from the set $T(L^c)^c$, the words having eligible subwords for replacements (that do not violate the containment constraint). For this, consider the set $(\Delta \cup \{m\})^* M (\Delta \cup \{m\})^*$. This is the set of all the words on $\Delta \cup \{m\}$ that have at least one *potentially* eligible subword for replacement. But, as said before, this replacement should be "contained" in the language L. Formally speaking, we are interested in solving the following language equation. Find the biggest languages $X, Y \subseteq (\Delta \cup \{m\})^*$ such that

$$XMY \subseteq (T(L^c))^c.$$

Then, if we are able to find such X and Y, we have that XMY is the set of *all* the words on $\Delta \cup \{m\}$, that still have eligible subwords for replacement. Clearly, we filter out such words and have that

$$MPR_M(L) = (T(L^c))^c \cap (XMY)^c.$$

So, let's examine how to solve the above language equation. For this, consider another special symbol m', such that $m' \notin \Delta \cup \{m\}$, and the substitution def' : $\Delta \cup \{m\} \cup \{m'\} \longrightarrow \Delta \cup \{m\}$, such that

$$def'(m') = M, \;\; def'(m) = m, \text{ and } \; def'(R) = R \text{ for } R \in \Delta.$$

Now, we build a transducer T' in the same way as we did for the transducer T, but for the extended alphabet $\Delta \cup \{m\}$ and considering m' as the special symbol. Reasoning similarly as before, if we transduce the complement of $(T(L^c))^c$ i.e. $T(L^c)$, we have that $(T'(T(L^c)))^c$ is the set of *all* words $W \in \Delta \cup \{m\} \cup \{m'\}$, such that $def'(W) \subseteq (T(L^c))^c$. However, in order to solve the above language equation, we will be interested in the subset of words in $(T'(T(L^c)))^c$ that contain exactly *one* special symbol m'. To get this subset, we filter out the un-wanted words by intersecting as in the following

$$(T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*.$$

The above is the set of *all* words $W = W_1 m' W_2$, where $W_1, W_2 \in (\Delta \cup \{m\})^*$, and $def'(W) \subseteq (T(L^c))^c$. Consider now the transducers T_l and T_r that erase from a word $W \in (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*$ the subword in $(\Delta \cup \{m\})^* m'$ and $m'(\Delta \cup \{m\})^*$ respectively. Finally, we set

$$X = T_r((T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*),$$

and

$$Y = T_l((T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*).$$

It is easy to see now that X and Y are regular languages and the maximal solution to the above equation. □

As a generalization of Theorem 4, we can give the following result about the maximal partial $\mathbf{V}$-rewriting, of a query Q with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

Theorem 5. *Given a regular path query Q, the $MPR_{\mathbf{V}}(Q)$ can be effectively computed.*

Proof. Let $A_i = (S_i, \Delta, \delta_i, s_{0i}, F_i)$, for $i \in [1, n]$, be n nondeterministic finite automata that accept the corresponding V_i languages. Let us consider the finite transducer:

$$T = (S_1 \cup \ldots \cup S_n \cup \{s_0'\}, \Delta, \Delta \cup \Omega, \delta', s_0', \{s_0'\}),$$

where

$$\begin{aligned}\delta' = & \{(s, R, s', \epsilon) : (s, R, s') \in \delta_i,\ i \in [1, n]\} \cup \\ & \{(s'_0, R, s'_0, R) : R \in \Delta\} \cup \\ & \{(s'_0, R, s, \epsilon) : (s_{0i}, R, s) \in \delta_i,\ i \in [1, n]\} \cup \\ & \{(s'_0, R, s'_0, v_i) : (s_{0i}, R, s) \in \delta_i \text{ and } s \in F_i,\ i \in [1, n]\} \cup \\ & \{(s, R, s'_0, v_i) : (s, R, s') \in \delta_i \text{ and } s' \in F_i,\ i \in [1, n]\}.\end{aligned}$$

The transducer T performs the following task: given a language L' as input, it produces as output all the words of L', having replaced in them an arbitrary number of words from $V_1 \cup \ldots \cup V_n$ with the corresponding special symbols.

Let's now consider the transduction $T(Q^c)$. As in the proof of Theorem 4, we can also see the transduction $T(Q^c)$ as the set of all words W on $\Delta \cup \Omega$ such that $def(W) \cap Q^c \neq \emptyset$. Hence, $(T\,(Q^c))^c$, being the complement of this set, will contain only $\Delta \cup \Omega$ words, such that *all* the Δ-words in their substitution by *def* will be contained in Q. This is the containment condition for a word on $\Delta \cup \Omega$ to be in the $MPR_{\mathbf{V}}(Q)$.

However, what we like is not a contained arbitrary, but a contained exhaustive replacement of words from $V_1 \cup \ldots \cup V_n$. To achieve this goal, we filter out from $T(Q^c)^c$ the words that still have subwords for potential "contained replacements." For this, we solve the language equations as in Theorem 4, i.e. find the biggest languages $X_i, Y_i \subseteq (\Delta \cup \Omega)^*$ such that

$$X_i V_i Y_i \subseteq (T(Q^c))^c,$$

for $i \in [1, n]$. Then, the $MPR_{\mathbf{V}}(Q)$ is obtained by filtering $T(Q^c)^c$ as in the following

$$T(Q^c)^c \bigcap \left(\bigcup_{i=1}^{n} X_i V_i Y_i \right)^c.$$

□

5 Optimizing Regular Path Queries Using Partial Rewritings

In this section we show how to utilize partial rewritings in query optimization in a scenario where we have available a set of precomputed views, as well as the database itself. The views could be materialized views in a warehouse, or locally cached results from previous queries in a client/server environment. In this scenario, the views are assumed to be exact, and we are interested in answering the query by consulting the views as far as possible, and by accessing the database only when necessary.

Formally, let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of view definitions, and let $\Omega = \{v_1, \ldots, v_n\}$ be the view alphabet as before. We define the *view-graph* $\mathcal{V}$ to be a database over (D, Ω) induced by the set

$$\bigcup_{i \in \{1,\ldots,n\}} \{(a, v_i, b) : (a, b) \in ans(V_i, DB)\}.$$

of Ω-labeled edges.

It has been shown in [4] that, given a query Q, if the $MCR_{\mathbf{V}}(Q)$ is exact, then $ans(Q, DB) = ans(MCR_{\mathbf{V}}(Q), \mathcal{V})$. However, the cases when we are able to obtain an exact complete rewriting of the query using the views could be rare in practice. In general, we have in the views only part of the information needed to answer the query. In the following, we will use exact partial rewritings not to *completely* avoid accessing the database, but to *minimize* such access as much as possible. We can use an exact partial rewriting Q' to evaluate the query on the view-graph, accessing the database in a "lazy" fashion only when necessary.

The intuition behind our algorithm is that we use the view-graph as it was the database, and we extend it "on demand" as the navigation, guided by an automaton for Q', requires. In fact, if we need to access a node in the database, we add to the view graph *all* (not only what we need) the outgoing edges and the neighboring nodes. This was motivated by considering the database as a set of HTML (or XML) pages related to each other through links. Accessing a page in the web could be time consuming, but subsequently, parsing the page text in the main memory for finding the links and the addresses, where these links point, is efficient. Since our queries could be recursive, a page (node) could be visited many times, and if it has already been fetched before from the database, along with all of its links, then there is no need to consult the database again. So, during the execution of the algorithm, for each page x that we fetch from the database, we create a flag $Expanded_x$ and set it to true, meaning that now we have full local information for this page.

Algorithm 1
Input: An exact partial rewriting Q' for a query Q and a database DB.
Output: The answer to the query Q on database DB.
Method: First construct an automaton $A_{Q'}$ for Q'. Let s_0 be the initial state in $A_{Q'}$. Then, for each node $a \in N$, we compute a set $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(a, s_0)\}$.
2. For each transition $s_0 \xrightarrow{R} s$ in $A_{Q'}$, access DB and add to $\mathcal{V}$ the subgraph of DB induced by *all* the edges originating from the nodes b, which have some outgoing edges R. For each such node b, create a flag $Expanded_b$, and set it to **true**.
3. Repeat 4 until $Reach_a$ no longer changes.
4. Choose a pair $(b, s) \in Reach_a$.
   (a) If there is a transition $s \xrightarrow{v_i} s'$ in $A_{Q'}$, and there is an edge $b \xrightarrow{v_i} b'$ in $\mathcal{V}$, then add the pair (b', s') to $Reach_a$.
   (b) Similarly, if there is a transition $s \xrightarrow{R} s'$ in $A_{Q'}$, and there is an edge $b \xrightarrow{R} b'$ in $\mathcal{V}$, then add the pair (b', s') to $Reach_a$.
   (c) Otherwise, if there is a transition $s \xrightarrow{R} s'$ in $A_{Q'}$, but there is not an edge $b \xrightarrow{R} b'$ in $\mathcal{V}$, and there is not a flag $Expanded_b = \mathbf{true}$, then access the database and add to $\mathcal{V}$ the subgraph of DB induced by *all* the edges originating from b. Create a flag $Expanded_b$, and set it to **true**.

Set $eval(Q', \mathcal{V}, DB) = \{(a, b) : (b, s) \in Reach_a$, and s is a final state in $A_{Q'}\}$. □

It is easy to see that the following theorem is true.

Theorem 6. *Given a query Q and a set $\mathcal{V}$ of exact cached views, using an exact partial rewriting Q', we have that*

$$eval(Q', \mathcal{V}, DB) = ans(Q, DB).$$

6 Optimizing Conjunctive Regular Path Queries

A *conjunctive regular path query* (CRPQ) Q is an expression of the form

$$Q(x_1, \ldots, x_l) : -y_1 E_1 z_1, \ldots, y_k E_k z_k,$$

where $x_1, \ldots, x_l$, $y_1, \ldots, y_k$, $z_1, \ldots, z_k$ are (not necessarily all distinct) variables over the universe of objects D, such that all the distinguished (head) variables $x_1, \ldots, x_l$ occur in the body, i. e. each x_i is some y or z, and $E_1, \ldots, E_k$ are regular languages (or regular path queries, RPQ) over the database alphabet Δ. We call the conjunctions yEz of a CRPQ, *regular path atoms*, or simply *atoms*.

The answer set $ans(Q, DB)$ to a CRPQ Q over a database $DB = (N, E)$ is the set of tuples $(a_1, \ldots, a_l)$ of nodes of DB, such that there is a total mapping τ from $y_1, \ldots, y_k$, $z_1, \ldots, z_k$ to N with $\tau(x_i) = a_i$ for every distinguished variable x_i of Q, and $(\tau(y), \tau(z)) \in ans(E, DB)$ for every atom yEz in Q.

We say for an atom yEz, when $\tau(y) = a$ and $\tau(z) = b$, that y and z have been *bound* to the objects a and b respectively.

Suppose now, that we have available a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of conjunctive regular path views, which are defined as

$$V_i(x_{i1}, \ldots, x_{il_i}) : -y_{i1} E_{i1} z_{i1}, \ldots, y_{ik_i} E_{ik_i} z_{ik_i},$$

for $i \in [1, n]$. Let $\mathbf{E}$ be the set of all E_{ij} above, for $i \in [1, n]$, and Ω be a set of $e_{ij} \notin \Delta$ corresponding symbols. Then, we define the *conjunctive exact partial* $\mathbf{V}$-*rewriting*, or $CEPR_{\mathbf{V}}(Q)$, of a CRPQ Q with respect to a set $\mathbf{V}$ of conjunctive regular path view definitions, as

$$CEPR_{\mathbf{V}}(x_1, \ldots, x_l) : -y_1 E'_1 z_1, \ldots, y_k E'_k z_k,$$

where E'_i, for $i \in [1, k]$, is an *exact* partial rewriting of the regular language E_i with respect to $\mathbf{E}$, with Ω as the corresponding set of special symbols.

Suppose that the sub-mechanism for answering the regular path atoms of the CRPQ's remembers the regular expressions and caches the corresponding answer sets of the regular path atoms in the recently processed CRPQ's. Observe that the cached answer sets of the regular path atoms E_{ij} do not necessarily contain the full set $ans(E_{ij}, DB)$, but only those pairs that were called for, by the sideways information passing mechanism used. To formalize the "fullness" of a cached answer set, we introduce the notions of the "global completeness" and "local completeness," for the subsets of $ans(E, DB)$, for some E.

Let P be a subset of $ans(E, DB)$, for some E and DB. Then P is said to be *globally complete* if $P = ans(E, DB)$. On the other hand, P is *locally complete with respect to a node* a, if $P \supseteq \sigma_{\$1=a}(ans(E, DB))$. If P is locally complete wrt all nodes in $\pi_{\$1}(P)$, we say that P is *locally complete* [2].

We observe that, if the query processor can traverse the database graph only in the forward direction (as is the case in the web, see [2]), then the cached answer set of any atom $yE_{ij}z$, for which z is not already bound to some objects, is locally complete. For simplicity, we call the atoms *locally complete*, when we have computed for them locally complete answer sets. As we will see, the locally complete atoms can be very useful in the query optimization.

Now, let's consider the atoms, which have the z variable already bound to some objects. We observe that, for such atoms, we could have computed locally incomplete answer sets. In order to see this, suppose for example, that the variable z has been bound to the objects b and c. Then, if the variable y bounds to an object, say a, the RPQ answering sub-mechanism using "cut"-like constructs, can stop the evaluation after computing in the answer, starting to navigate from a, the objects b and c. But, the object a could be connected with paths spelling words in the same regular language, to other database objects besides b and c.

However, if the variable z has been also bound to another object, say d, which cannot be reached from a, by following a path spelling a word in E_{ij}, then we inevitably have to compute a locally complete answer set for the (sub)atom $aE_{ij}z$, in order to reject the pair (a, d). I such a case, the "cut"-like constructs do not have have a chance to execute. So, $aE_{ij}z$ is locally complete, and we cache its answer set for future optimizations.

Now, let's see how we can optimize the answering of the conjunctive regular path queries using conjunctive exact partial rewritings.

Consider the regular path atom $yE_{ij}z$ in some recently processed view V_i. As explained before, if the variable z is not bound to some database object, then $yE_{ij}z$ is locally complete. Otherwise, we consider all the (sub)atoms $aE_{ij}z$, which are locally complete. Let's denote with $cache_{ij}$, the cached answer set of the locally complete atom $yE_{ij}z$, or if otherwise, the union of the cached answer sets of its locally complete (sub)atoms. Then, we define the *atom-view-graph* $\mathcal{V}$ to be a database over (D, Ω) induced by the set

$$\bigcup_{i \in \{1,\dots,n\}} \{(a, e_{ij}, b) : (a, b) \in cache_{ij}\}.$$

of Ω-labeled edges. [3]

The evaluation will alternate between the database-graph and atom-view-graph. Suppose that we want to answer the atom yEz and y has already been bound to a node, say a. We now need to compute the nodes reachable from a, by

[2] We consider for the simplicity of notations a set of object pairs as a binary relational table.

[3] From another point of view, we can also see that a regular path atom (or (sub)atom), along with its locally complete answer set, is nothing else but a local node constraint as defined in [2].

a path spelling a word in E. Recall that for E we have computed a corresponding exact rewriting E'. We start by answering $aE'z$ on DB. Intuitively, when during the navigation we are in a node b, and the regular expression for E' requires an Ω-symbol, say e_{ij}, to be matched, then we look at the atom-view-graph $\mathcal{V}$. If there is a node b in $\mathcal{V}$, and in that node we find outgoing e_{ij} edges, then we advance the navigation in the atom-view-graph. Since the cached answer for E_{ij} is locally complete, we are sure that we get all the answers had we answered E_{ij} directly in the database starting from b.

On the other hand, if we do not find an object b in $\mathcal{V}$, or even if we find b in $\mathcal{V}$, but there are no outgoing edges labeled with e_{ij}, then we evaluate E_{ij}, starting from b in the database graph, and enrich the atom-view-graph accordingly. Clearly, we do not add by this operation any overhead for answering parts of some new Δ^* word, that is not in E, because the rewriting is exact. Formally, we have the following algorithm for answering aEz on the basis of E'.

Algorithm 2

Input: An exact rewriting E' for the (sub)atom aEz, and a database DB.

Output: The answer to the (sub)atom aEz on database DB.

Method: First construct an automaton $A_{E'}$ for E'. Let s_0 be the initial state in $A_{E'}$. We compute the set $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(a, s_0)\}$.
2. Repeat 3 until $Reach_a$ no longer changes.
3. Choose a pair $(b, s) \in Reach_a$.
   (a) If there is a transition $s \xrightarrow{e_{ij}} s'$ in $A_{E'}$, and there is an edge $b \xrightarrow{e_{ij}} b'$ in $\mathcal{V}$, then add the pair (b', s') to $Reach_a$.
   (b) Otherwise, if there is a transition $s \xrightarrow{e_{ij}} s'$ in $A_{E'}$, but there is not an edge $b \xrightarrow{e_{ij}} b'$ in $\mathcal{V}$, answer $bE_{ij}z$ on DB and enrich $\mathcal{V}$ accordingly.
   (c) If there is a transition $s \xrightarrow{R} s'$ in $A_{E'}$, and there is an edge $b \xrightarrow{R} b'$ in the database DB, then add the pair (b', s') to $Reach_a$.

Finally, set $eval(E', a, DB) = \{(a, b) : (b, s) \in Reach_a, \text{ and } s \text{ is a final state in } A\}$. □

It is easy to see that the following theorem is true.

Theorem 7. *Given a CRPQ Q and a set $\mathbf{V}$ of views as above, using the rewriting $CEPR_{\mathbf{V}}(Q)$, we have that for some regular atom E in Q, the answer to aEz equals $eval(E', a, DB)$.*

Now, for the case of regular path atoms aEz, where the variable z has already been bound to some objects, we slightly modify the above algorithm to evaluate the answer to such atoms as well. For this, let B be the set of objects where z has been bound. Then, in the above algorithm we *incrementally* compute in each step the set $eval(E', a, DB)$ and change the loop (2) to

Repeat 3 until $Reach_a$ no longer changes or $\pi_{\$2}(eval(E', a, DB))$ equals B.

We note that, the second condition of the loop termination is there for restricting the search space, in case we find (or verify) that we can reach from a *all* the objects in B by following paths, which spell words in E. Clearly, in this case, the answer set of the atom aEz equals $a \times B$. On the other hand, we also stop if $Reach_a$ reaches a fixed point, and in this case the answer to the atom aEz equals $a \times (\pi_{\$2}(eval(E', a, DB)) \cap B)$. In such a case, although the variable z had already been bound, the set $eval(E', a, DB))$ is fully computed and so, it is locally complete.

Finally, if we set $B = \emptyset$, when the variable z has not been already bound to some objects, then we can have a single *RPQ-answer-and-cache* algorithm. This algorithm would compute the answer set to an atom yEz, by iterating over all the objects a, where the variable y could be bound, and compute aEz by using an exact partial rewriting $aE'z$, as described above. At the end, we check whether the set $Reach_a$ has reached a fixed point. If yes, then we cache the locally complete $eval(E', a, DB))$, for future RPQ optimizations.

7 Complexity Analysis

Theorem 8. *Given a language L and an ϵ-free language M both of them over an alphabet Δ, the problem of generating the $MPR_M(L)$ is in 3EXPTIME.*

Proof. Let us refer to the constructive proof of the Theorem 4. From there we have that $MPR_M(L) = (T(L^c))^c \cap (XMY)^c$, where $X = T_r((T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*)$ and $Y = T_l((T'(T(L^c)))^c \cap (\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*)$.

Consider the first term of the intersection. To compute the complement Q^c of the query is exponential. To transduce it to $T(Q^c)$ is polynomial. To complement again is exponential. So, we can compute the first term in 2EXPTIME.

Now, let's consider the second term. To compute $T'(T(L^c)))^c$, similarly as before, is 2EXPTIME. To compute and then intersect with $(\Delta \cup \{m\})^* m' (\Delta \cup \{m\})^*$ is polynomial. To transduce with T_r or T_l is polynomial. So, to compute XMY is 2EXPTIME.

Finally, to complement XMY is 3EXPTIME and so, the generation of the $MPR_M(L)$ is 3EXPTIME + 2EXPTIME = 3EXPTIME. □

From the above theorem, we can easily derive the following corollary, regarding the upper complexity bound for the generation of the $MPR_{\mathbf{V}}(Q)$ of a query Q, with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

Corollary 3. *Given a language Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the problem of generating the $MPR_{\mathbf{V}}(Q)$ is in 3EXPTIME.*

We emphasize here that the above complexity analysis is a worst case analysis. In fact, the above complexity comes from possible DFA's state "blow up." However, despite these worst case possibilities, experimental results in [8], for converting graphs into DFA's, are encouraging, indicating that for the majority of the cases, the running time is very reasonable and the resulting DFA's are significantly smaller than their sources. Also, observe that if the probability of getting an exponential size DFA for an NFA is $p << 1$, then the probability of getting a triple exponential "blow up" in our case would roughly be p^3, which is very small.

Now, in order to prove that the above established upper bound is essentially optimal we will need the following constructions and theorems. Consider the set

$$(T(Q^c))^c \cap \left((\Delta \cup \Omega)^* (V_1 \cup \ldots \cup V_n)(\Delta \cup \Omega)^*\right)^c,$$

where the transducer T is defined as in Theorem 5. This is in essence the *exhaustive lower partial rewriting*, or $ELPR_{\mathbf{V}}(Q)$ of [11], which is also discussed in Section 3. It is the set of all "mixed" words W on the alphabet $\Omega \cup \Delta$, with no subword in $V_1 \cup \cdots \cup V_n$, such that their substitution by *def* is contained in the query Q. Obviously, $def(ELPR_{\mathbf{V}}(Q)) \subseteq Q$ and we are interested in the complexity of testing its exactness with respect to the query Q [4]. We prove the following theorem regarding the upper bound of the above exactness problem.

Theorem 9. *Given a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the problem of testing $def(ELPR_{\mathbf{V}}(Q)) = Q$ is in 2EXPSPACE.*

Proof. By Theorem 8 the automaton $(T(Q^c))^c$ can exponentially "blow up" and namely be of doubly exponential size. The automaton for

$$\left((\Delta \cup \Omega)^* (V_1 \cup \ldots \cup V_n)(\Delta \cup \Omega)^*\right)^c$$

can also exponentially "blow up" but its size can be up to single exponential. So, in total the automaton for $ELPR_{\mathbf{V}}(Q)$ can be of up to doubly exponential size. Now, let's consider $def(ELPR_{\mathbf{V}}(Q))$. For the substitution *def* there exists a polynomial size transducer T_{def} such that $def(ELPR_{\mathbf{V}}(Q)) = T_{def}(ELPR_{\mathbf{V}}(Q))$ [18, 10]. Thus, the size of the automaton $T_{def}(ELPR_{\mathbf{V}}(Q))$ will be polynomial on the size of the automaton for $ELPR_{\mathbf{V}}(Q)$ i.e. it can be doubly exponential on the size of the automaton for Q. Now, to test the exactness $def(ELPR_{\mathbf{V}}(Q)) = Q$ is in PSPACE with regard to the size of the automaton for $ELPR_{\mathbf{V}}(Q)$. So, in total we have that testing the exactness of $ELPR_{\mathbf{V}}(Q)$ is in 2EXPSPACE. □

For the lower bound of the exactness $def(ELPR_{\mathbf{V}}(Q)) = Q$ we will use the maximally contained rewriting, $MCR_{\mathbf{V}}(Q)$ of [3]. As mentioned in Section 2, the $MCR_{\mathbf{V}}(Q)$ is the set of *all* words W in Ω^*, such that $def(W) \in Q$. In [3], the exactness problem for the $MCR_{\mathbf{V}}(Q)$ is proven to be 2EXPSPACE complete. We will give in the following a reduction of the exactness problem for $MCR_{\mathbf{V}}(Q)$ to the exactness problem for $ELPR_{\mathbf{V}}(Q)$.

Consider a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions. Now, consider the set $\mathbf{V}' = \{V_1', \ldots, V_n'\}$ of new view definitions, where $V_i' = \$V_i\$$ for

[4] The complexity bounds for testing the exactness $ELPR_{\mathbf{V}}(Q)$ are not discussed in [11].

$i = [1, n]$ and $\$ \notin \Delta \cup \Omega$. Regarding the query Q, we will transform it into a new query Q' through a transduction. For this we construct the following transducer.

Let $A_i = (S_i, \Delta, \delta_i, s_{0i}, F_i)$, for $i \in [1, n]$ be n nondeterministic finite automata that accept the corresponding V_i languages. Let us consider the finite transducer: $T_{\$\$} = (S_1 \cup \ldots \cup S_n \cup \{s'_0\}, \Delta, \Delta \cup \{\$\}, \delta', s'_0, \{s'_0\})$, where

$$\begin{aligned}\delta' = &\{(s, R, s', R) : (s, R, s') \in \delta_i,\ i \in [1, n]\} \cup \\ &\{(s'_0, \epsilon, s_{0i}, \$) : i \in [1, n]\} \cup \\ &\{(s, \epsilon, s'_0, \$) : s \in F_i,\ i \in [1, n]\}.\end{aligned}$$

The transducer $T_{\$\$}$ performs the following task: given a language L as input, it produces as output all the words of L having inserted in them the symbol \$ to mark the beginning and the end of a subword that belongs to V_i for some $i \in [1, n]$. Moreover, only the words of L, which we can divide into a sequence of subwords, such that each belongs to some view language, are transduced. All the other L words are filtered out. Formally,

$$\begin{aligned}T_{\$\$}(L) = \{\$U_1\$\$U_2\$ \ldots \$U_k\$:\ & \\ &\text{for some } U \text{ in } L, U = U_1U_2 \ldots U_k \\ &\text{and } \forall i \in [1, k]\ \exists j \in [1, n] \text{ such that } U_i \in V_j\}.\end{aligned}$$

Theorem 10. *Consider a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions. Construct Q and $\mathbf{V}' = \{V'_1, \ldots, V'_n\}$ as above. Let $\Omega' = \{v'_1, \ldots, v'_n\}$ and def' be the usual view substitution on Ω', i.e. $def'(v'_i) = V'_i$, for $i \in [1, n]$. Also, let $def_{\$\to\epsilon}$ be the substitution that erases the symbol \$ from some language on $\Delta \cup \{\$\}$. Then, the following is true: the $MCR_{\mathbf{V}}(Q)$ is exact if and only if*

1. *$def_{\$\to\epsilon}(Q') = Q$, and*
2. *$ELPR_{\mathbf{V}'}(Q')$, with respect to $\mathbf{V}' = \{V'_1, \ldots, V'_n\}$, is exact.*

Proof. "if." The fact that $def_{\$\to\epsilon}(Q') = Q$ means that for *any* word $W \in Q$ there exists a representation $W = U_1U_2 \ldots U_k$ such that $\forall i \in [1, k]\ \exists j \in [1, n]$ for which we have $U_i \in V_j$. In other words, for each $W \in Q$ there exists a corresponding $W' \in Q'$ with inserted markers \$. Now, observe that, because of the markers \$, the $ELPR_{\mathbf{V}'}(Q')$ is forced to be equal with the $MCR_{\mathbf{V}'}(Q')$. So, the exactness of $ELPR_{\mathbf{V}'}(Q')$ means exactness of $MCR_{\mathbf{V}'}(Q')$, which in turn means that for each word $W' \in Q'$ there is a word $v'_{i_1} \ldots v'_{i_m}$ such that $W' \in def'(v'_{i_1} \ldots v'_{i_m})$, and $def'(v'_{i_1} \ldots v'_{i_m}) \subseteq Q'$. It is easy to see that for the corresponding word $W = def_{\$\to\epsilon}(W')$ we have that $W \in def(v_{i_1} \ldots v_{i_m})$ and $def(v_{i_1} \ldots v_{i_m}) \subseteq Q$. Since we showed that for each $W \in Q$ there exists a $W' \in Q'$, we imply that each word of Q can be represented by some word in Ω^* which belongs to the $MCR_{\mathbf{V}}(Q)$. Together with the fact that $def(MCR_{\mathbf{V}}(Q)) \subseteq Q$, from the above we conclude the exactness of $MCR_{\mathbf{V}}(Q)$.

"only if." From the exactness $def(MCR_{\mathbf{V}}(Q)) = Q$, we have that for each word $W \in Q$ there exists a word $v_{i_1} \ldots v_{i_m} \in MCR_{\mathbf{V}}(Q)$ such that $W \in def(v_{i_1} \ldots v_{i_m})$. But, this means that W can be written as $W = U_1 \ldots U_m$, where

$U_i \in V_i$, for $i \in [1, m]$. From this, we have that $W' = \$U_1\$\$U_2\$\ldots\$U_m\$ \in Q'$. Since all the above hold for each word of Q, the condition (1) follows.

Regarding the condition (2) recall that because of the markers \$ we have that $ELPR_{\mathbf{V}'}(Q') = MCR_{\mathbf{V}'}(Q')$. From the construction of Q', observe that for each word $v_{i_1} \ldots v_{i_m} \in MCR_{\mathbf{V}}(Q)$ the corresponding word $v'_{i_1} \ldots v'_{i_m}$ is in the $MCR_{\mathbf{V}'}(Q')$. From this, it is not difficult to conclude that $MCR_{\mathbf{V}'}(Q')$, and in turn $ELPR_{\mathbf{V}'}(Q')$ are exact with respect to Q'. $\square$

Based on the above theorem we give the following theorem regarding the optimality of the construction given in Theorem 9 for testing the exactness of the ELPR of a query Q.

Theorem 11. *Given a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the problem of testing $def(ELPR_{\mathbf{V}}(Q)) = Q$ is 2EXPSPACE complete.*

Proof. For the upper bound we refer to Theorem 9. For the lower bound consider the reduction from the exactness problem for the MCR of an arbitrary query and an arbitrary set of view definition to the testing of the conditions (1) and (2) in Theorem 10. Observe that the automaton for the modified query of the first condition, since it is being obtained by a polynomial size transducer, is polynomial as well. This means in turn that to test the language equality of the first condition of Theorem 9 is in PSPACE. So, since the exactness problem for the MCR is 2EXPSPACE complete, the lower bound for the exactness problem of the ELPR is also 2EXPSPACE. $\square$

Finally, the following theorem shows the optimality of our algorithm for computing the MPR of a query using a set of view definitions.

Theorem 12. *The algorithm presented in Theorem 5 for computing the rewriting $MPR_{\mathbf{V}}(Q)$ of a query Q given a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, is essentially optimal.*

Proof. Consider the $ELPR_{\mathbf{V}}(Q)$. Recall that this is the set of all "mixed" words W on the alphabet $\Omega \cup \Delta$, with no subword in $V_1 \cup \cdots \cup V_n$, such that their substitution by *def* is contained in the query Q. So, such words qualify for inclusion in $MPR_{\mathbf{V}}(Q)$. Hence, we have that $ELPR_{\mathbf{V}}(Q) \subseteq \mathrm{CEPR}(Q)$. On the other hand, observe that if $ELPR_{\mathbf{V}}(Q)$ is *exact* then we must have $ELPR_{\mathbf{V}}(Q) = MPR_{\mathbf{V}}(Q)$. Now, we can test the exactness of $ELPR_{\mathbf{V}}(Q)$ by testing the condition $MPR_{\mathbf{V}}(Q) \subseteq ELPR_{\mathbf{V}}(Q)$, which is equivalent with the non emptiness of $MPR_{\mathbf{V}}(Q) \cap (ELPR_{\mathbf{V}}(Q))^c$. Since, by Theorem 9, we get a DFA for $ELPR_{\mathbf{V}}(Q)$, to complement does not add anything to the complexity of testing the emptiness of the above intersection. Now, if we were able to generate the $MPR_{\mathbf{V}}(Q)$ in, say, 2EXPTIME, and since we are able to generate the $ELPR_{\mathbf{V}}(Q)$ in 2EXPTIME, then we could test the exactness of $ELPR_{\mathbf{V}}(Q)$ in 2EXPTIME, which is impossible by Theorem 11, unless 2EXPTIME = 2EXPSPACE. $\square$

8 Conclusions

In this paper we have studied the problem of partial query rewriting using only algebraic view definitions in the case where both the query and the views are expressed as regular path queries. We introduced maximal partial rewritings, MPR's, that are guaranteed to be both maximal and exact. Previously proposed rewritings did not have this property. We have characterized the computational complexity of computing MPR's and showed that our algorithm for computing the MPR's are essentially optimal. Then we have given two algorithms for utilizing partial rewritings in the context of cached views. These algorithms use as much as possible the cached information of the views and access the secondary or remote memory in a "lazy" fashion only on demand.

References

1. S. Abiteboul, P. Buneman and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.
2. S. Abiteboul, V. Vianu. Regular Path Queries with Constraints. *Journal of Computing and System Sciences* 58(3) 1999, pp. 428-452
3. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. *Proc. of PODS* 1999, pp. 194-204.
4. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Answering Regular Path Queries Using Views. *Proc. of ICDE* 2000, pp. 389-398.
5. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. View-Based Query Processing for Regular Path Queries with Inverse. *Proc. of PODS* 2000, pp. 58-66.
6. D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. What is View-Based Query Rewriting? *Proc. of KRDB* 2000, pp. 17-27.
7. D. Florescu, A. Y. Levy, D. Suciu Query Containment for Conjunctive Queries with Regular Expressions *Proc. of PODS* 1998, pp. 139-148.
8. R. Goldman and J. Widom. DataGuides: Enabling query Formulation and Optimization in Semistructured Databases. *Proc. of VLDB* 1997 pp. 436-445.
9. G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. *Proc. of ICDT* 1999 pp. 332-347.
10. G. Grahne and A. Thomo. An Optimization Technique for Answering Regular Path Queries. *Proc. of WebDB* 2000.
11. G. Grahne and A. Thomo. Algebraic Rewritings for Optimizing Regular Path Queries. *Proc. of ICDT* 2001 pp. 301-315.
12. J. E. Hopcroft and J. D. Ullman *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley 1979.
13. L. Kari. *On Insertion and Deletion in Formal Languages.* Ph.D. Thesis, 1991, Department of Mathematics, University of Turku, Finland.
14. A. Y. Levy. *Answering queries using views: a survey.* Technical Report, Computer Science Dept., Washington Univ., 2000.
15. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava. Answering Queries Using Views. *Proc. of PODS* 1995, pp. 95-104.
16. A. O. Mendelzon and P. T. Wood, Finding Regular Simple Paths in Graph Databases. *SIAM J. Comp. 24:6,* (December 1995).
17. Y. Papakonstantinou, V. Vassalos. Query Rewriting for Semistructured Data. Proc. of SIGMOD 1999, pp. 455-466

18. S. Yu. Regular Languages. In: *Handbook of Formal Languages*. G. Rozenberg and A. Salomaa (Eds.). Springer Verlag 1997, pp. 41-110